



Report on the RNN Code
PD-Internship
Lerner Paul
06/06/2018

Some of this report was originally present in report #4. However, since it's prone to update unlike the rest of report #4, on April 19th I split the report in 2 : Literature review and Code. This is the Code one.

In parallel I read about online HaW analysis and thus wrote report #5.

Disambiguation : All the results were obtained on the *spiral* task if not specified otherwise.

Until April 25th, all experiment results were obtained while validating on the test set (see [5 Evaluation and experiments](#) for discussion). Therefore, I added "[Validation accuracy !]" after all these results. Moreover, you'll see on most of the plots that there's only two curves : "train" and "validation", this is because there was no proper test set.

This report was previously named Report on the Code but I renamed it to RNN code as I started working on CNNs, although it doesn't contains only information about RNNs.

Cf. Git Commits and the experiment results.

Overview

1 Model architecture	2
2 Model training	7
2.1 Gradient clipping	7
2.2 Learning rate	8
2.3 Continuous learning	8
2.4 Training on validation set after early stopping	9
2.5 Batch training	10
3 Regularization	10
4 Data	11
4.1 Exploration	11
4.2 Preprocessing	11
4.3 Representation	13

4.4 Augmentation	14
4.4.1 Flips	15
4.4.2 Rotations	15
4.4.3 Translations	16
4.4.4 Homothetic (i.e. "zoom")	16
4.4.5 Gaussian noise	16
4.5 NewHandPD	18
5 Evaluation and experiments	19
6 Visualization & Interpretation	22
6.1 Input weights	22
6.2 Forget Gate	22
6.3 Misclassified subjects	22
Conclusion - Todo List	24
References	25

1 Model architecture

We use the LSTM as defined in PyTorch¹, thus, for each element in the input sequence, each LSTM unit of each layer computes the following operations :

$$\begin{aligned}
 i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\
 f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\
 g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \\
 o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\
 c_t &= f_t c_{(t-1)} + i_t g_t \\
 h_t &= o_t \tanh(c_t)
 \end{aligned}$$

Where h_t is the hidden state at time t , c_t is the cell state at time t , x_t is the input at time t , $h_{(t-1)}$ is the hidden state of the layer at time $t-1$, and i_t , f_t , g_t , o_t are the input, forget, cell, and output gates, respectively. σ is the sigmoid function. W_* are the weight matrices and b_* are the bias vectors.

Baseline decoder : We currently feed the last hidden state of the LSTM (summed if bLSTM) to a FC layer with a single neuron which is used for binary classification after applying a Sigmoid to it. The Sigmoid function is an activation function which is defined as :

$$f(x) = \frac{1}{1 + e^{-x}}$$

¹ <https://pytorch.org/docs/stable/nn.html#torch.nn.LSTM>

The input is a time-varying sequence (2k timesteps in average) where each timestep is a vector containing 7 measures : y-coordinate, x-coordinate, timestamp, button_status, tilt, elevation, pressure.

We train the model using Adam optimizer (Kingma & Ba) which is based on SGD. The optimizer minimizes the Binary Cross Entropy (BCE) Loss, which is defined as :

$$BCE = y \log(f(\hat{y})) - (1 - y) (\log (1 - f(\hat{y})))$$

Where y is the ground truth, aka the target aka the label (i.e. 0 for control and 1 for PD) ; \hat{y} is the prediction of the model and f is the Sigmoid function, defined above.

Our dataset is challenging :

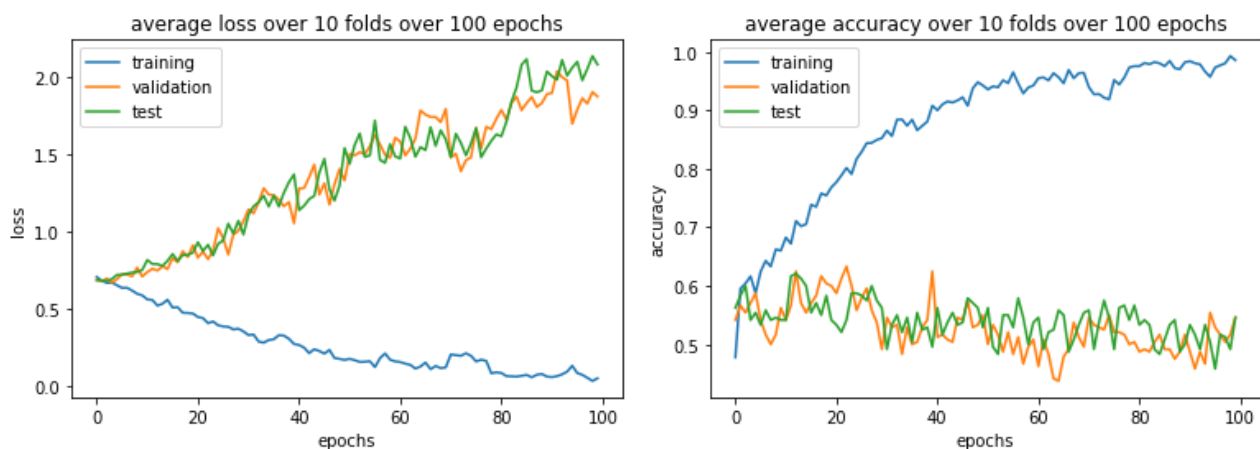
1. by its very small number of sample
2. by its very long and varying sequences

Thus our model should be complex enough to capture the long sequences but not too much or heavily regularized so it won't overfit the very little number of samples.

With a very simple model (**baseline**) :

- $learning_rate = 10^{-3}$
- $hidden_size=100$
- $num_layers=1$
- $bidirectional=False$
- $dropout=0.0$
- $no\ gradient\ clipping$
- → **43 701** total parameters, all trainable.

We're able to fit the training set (80% of the dataset), cf. figures below.



Although the model starts overfitting after 1, 2, 1, 4, 4, 2, 4, 3, 5 and 1 epochs (out of 10 folds, respectively).

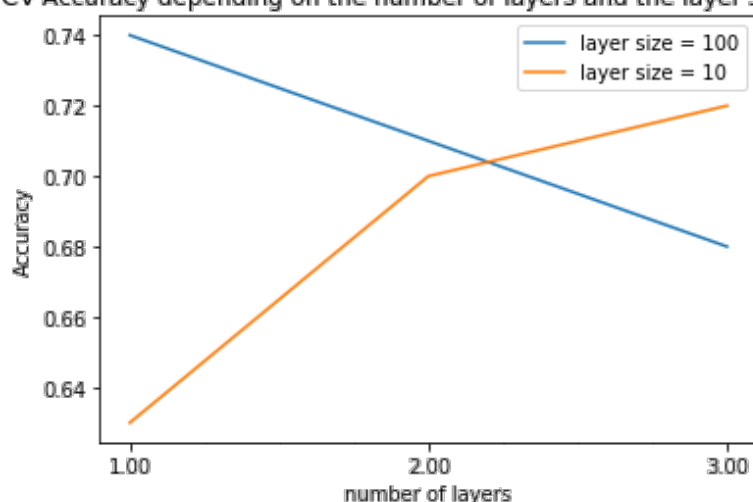
We don't use mini-batch as the sequences in our dataset are of very different lengths and we have not found a way of packing them efficiently.

Using a bidirectional LSTM consistently improved the results over the *spiral* task.

There is a lot of interaction between the layer size and the number of layers, we can observe a reversed effect, see figure below. The other hyperparameters are fixed to :

dowsampling factor	learning_rate	bidirectional	carry over	dropout	gradient clipping
1	0.001	TRUE	0	0.5	5

CV Accuracy depending on the number of layers and the layer's size



[Validation accuracy !]

This can probably be explained by the n° of

parameters :

layer's size	n° layers	n° of parameters
10	1	1531
10	2	4091
10	3	6651
100	1	87301
100	2	328901
100	3	570501

Therefore, the optimal n° of parameters of the network is probably between 6k and 90k, when using the current regularization techniques (i.e. 50% feedforward dropout). To confirm this hypothesis, I tried with *hidden_size=50 num_layers=1* → **23651** parameters and achieved similar results. However with *hidden_size=50 num_layers=2* → **84451** parameters the results were slightly inferior. Thus we should privilege the number of layers over the hidden size for the same numbers of parameters. Although this could be explained because of our

regularization techniques (i.e. 50% feedforward dropout), if we'd apply recurrent dropout the results might be different. **[Validation accuracy !]**

Among this similar results, I prefer to stick with the simpler (i.e. with less parameters) model which is less prone to overfitting :

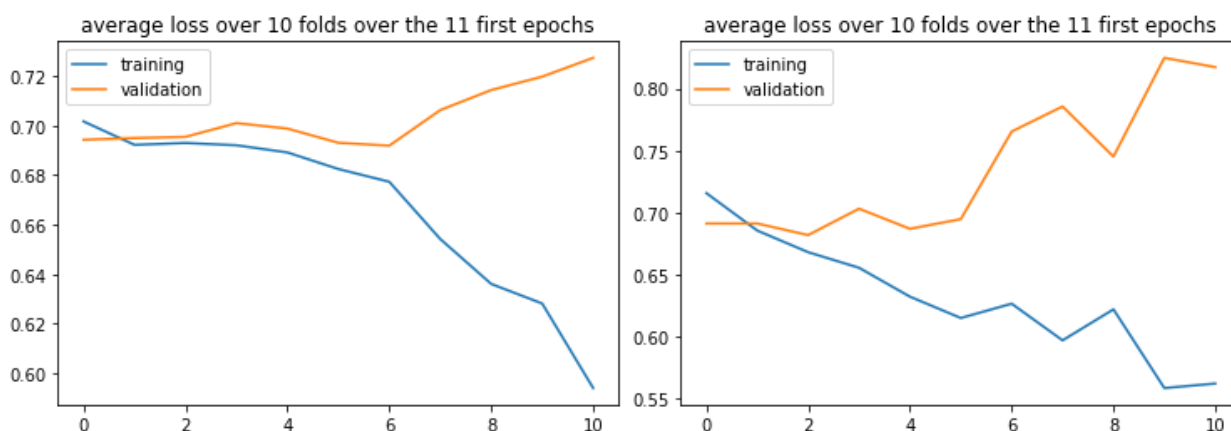
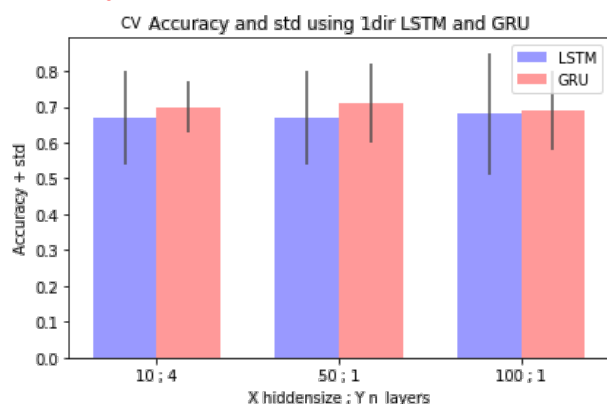


Fig. Left : *hidden_size=10 num_layers=3*, **right :** *hidden_size=100 num_layers=1*. **Different scales**

Using a 1d GRU consistently improved results and decreased std over 1d LSTM **[Validation accuracy !]** :



I found this first GRU model of 4 layers to be quite efficient, and ran several experiments with it. I call it GRU-baseline.

I performed a random search on the *l* task on :

- *learning_rate* in {0.01, 0.001, 0.0001}
- *num_layers* in {1, 2, 3, 4, 5}
- *hidden_size* in [[1, 200]]
- *bidirectional* in {True, False}
- *dropout* in [0, 1]

With an extra constraint : $num_layers * hidden_size < 300$ in order to limit the n° of parameters. The random search performed 59 experiments (10-CV) in 3 days. Unfortunately it didn't get better than my manual fine-tuning. We should be very careful of the cross validation results because of the early stopping. One set of parameters achieved 70% validation accuracy but had only 51% training accuracy so I had a look and the model was almost random (cf figure below). Almost all experiments results have better training than validation accuracy. Although this is partly explained by the early stopping (cf. [5 Evaluation and experiments](#)), it is also because we first train the model before validating so a more relevant comparison would be between validation accuracy at epoch *e* and training accuracy at epoch *e* + 1. Also, dropout is only active while training so the training accuracy should be higher than the validation one if there's dropout.

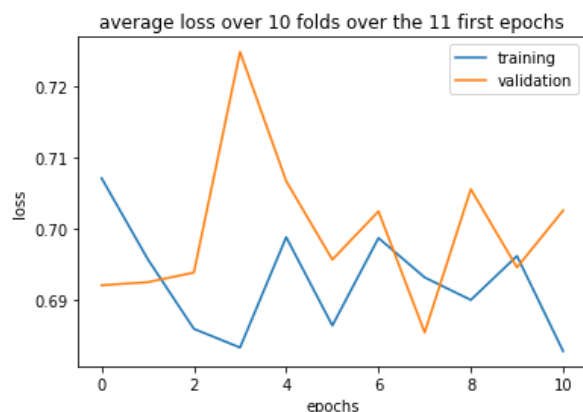



Fig. Model that doesn't fit but get's 70% accuracy on the / task

Following advices of Jozefowicz et al. 2015 (cf. report #4), we initialize the bias of the forget gate to 1 when using an LSTM as encoder.

It improved training accuracy, validation PPV and reduced the variance (std) on the small baseline model with 2 layers (colored in red in the experiments results).

Inspired by the work of Graves et al. on Hierarchical RNNs (cf. Report #4). I tried to implement a model where  different layer of a RNN (LSTM or GRU) work on different data scales. In order to do so I split the tasks into letters (I experimented with the / task). The first layer works on the timestep scale and gets fed the measure vector at each time step for the whole letter length. The second layer works on the letter scale and gets fed the last hidden state (i.e. letter encoding) of the first layer for each letter.

However the model hardly fits the data as you can see in the figure below. I've tried different learning rates, layer size, GRU and LSTM, downsampling the data... Update : this may be because the ls are not standardized (the whole task is standardized) thus further experiments are required.

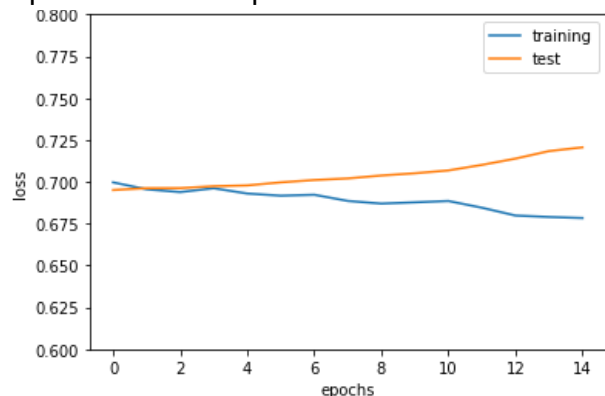


Fig. Hierarchical model hardly fits the data.

2 Model training

2.1 Gradient clipping

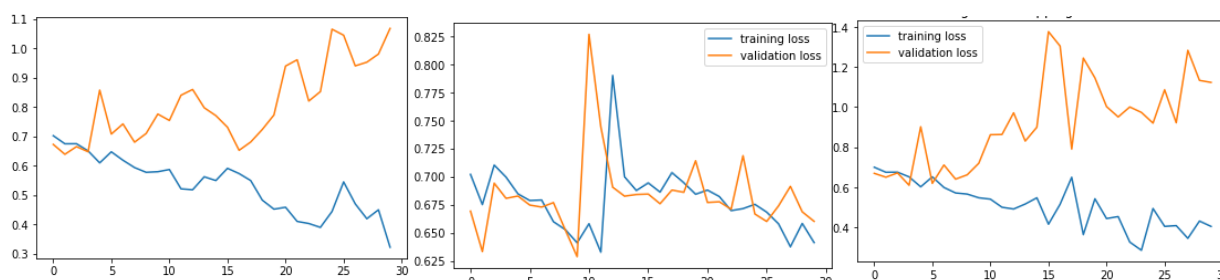


Fig. Baseline model (1 Model architecture) : on some folds the model won't fit the data very well (left : typical fold, middle : hard fold, right : clipped hard fold) /!\ Different scales /!

It may be because of exploding gradients : clipping the gradients norm to 5 helped with this (cf. figure).

Although it didn't help with the cross validation results, on the contrary, they are slightly inferior compared to the ones without gradient clipping. The differences between these results might also be explained by the random initialization of the weights of the network. I set the random seed after the first few experiments to avoid this bias.

Unlike with *spiral* above, clipping the gradients to 5 didn't have any effect on the *l* task (with $lr=10^{-3}$ or 10^{-4}).

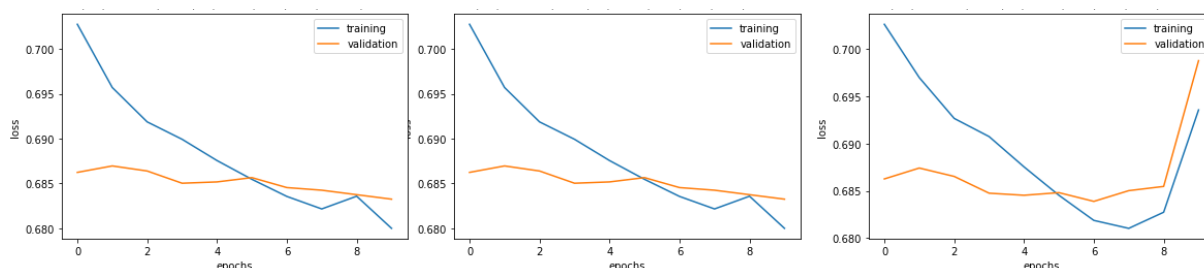


Fig. Left : no gradient clipping, middle : gradient clipping at 5, right : gradient clipping at 1.

Note : on these above experiments and until May 2nd, I used to clip both encoder's (i.e. LSTM or GRU) and decoder's (i.e. FC layer) gradients. Whereas gradient clipping should only be applied to RNNs which have the exploding gradients problem. Therefore, on May 2nd I tried to clip only the encoder's gradients and it didn't have any effect on the model :

task	model	learning rate	hidden size	num layers	bidirectional	dropout	gradient clipping
l	paper_air LSTMs, 1 decoder	0.001	10	4	FALSE	0	5

2.2 Learning rate

According to these first experiments, the best learning rate is 10^{-3} . With 10^{-4} there is more variance (std) and on some folds the RNN is overfitting from the first epoch (cf. figure below), we might try it again along with regularization techniques.

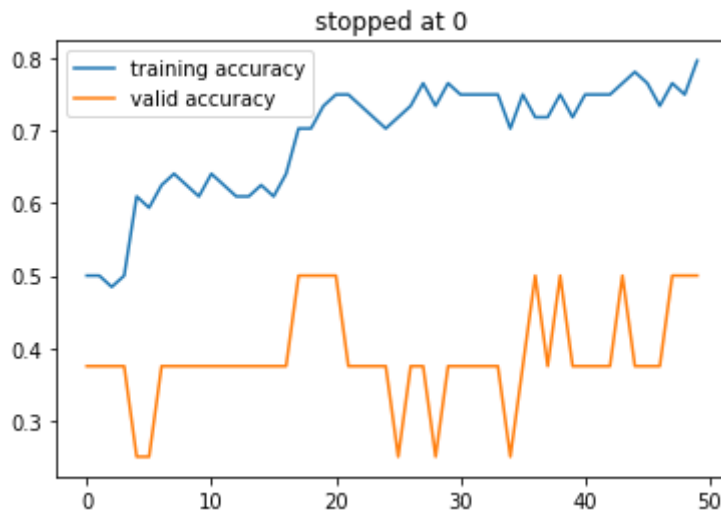


Fig. $LR = 10^{-4}$, RNN is overfitting from the first epoch


2.3 Continuous learning

I tried to train the model (GRU-baseline) "continuously" by "passing on" the hidden state from one subject to another (i.e. the hidden state is not init to 0 but to the previous hidden state). This makes the loss more unstable as you can see below. I highlighted the results in **purple** in the results summary. Although one can notice that the training loss is similar between the 2 conditions. It's quite strange given that there's no dropout so maybe the changes in the validation loss are not significant.

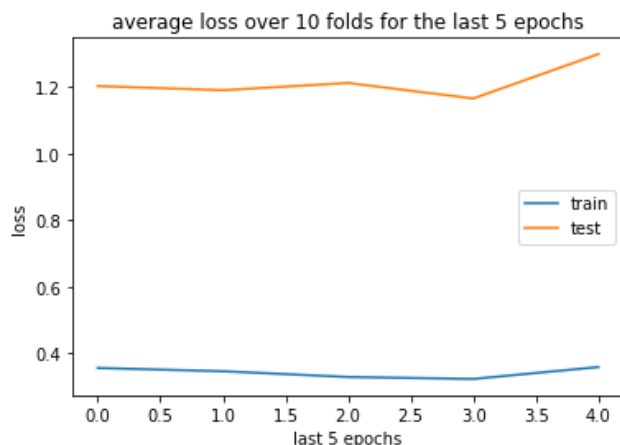


Fig. Continuous learning. Left : no, right : yes.

2.4 Training on validation set after early stopping

In order to train the data on a  larger dataset and still perform early stopping I've tried to train the model on both the training and validation set, once the early stopping criterion had been met (after reloading the best model). The results are not encouraging. See below the plot of

the average loss for the last 5 epochs (when re-training on validation set). We can see that the test loss has a tendency to grow.



3 Regularization

Consistently with Lipton et al., adding a dropout of 0.5 between the non-recurrent layers enhances the results with large layers and stacked bLSTMs. See figure below, the lr is 10^{-3} and gradient's norm is clipped at 5.

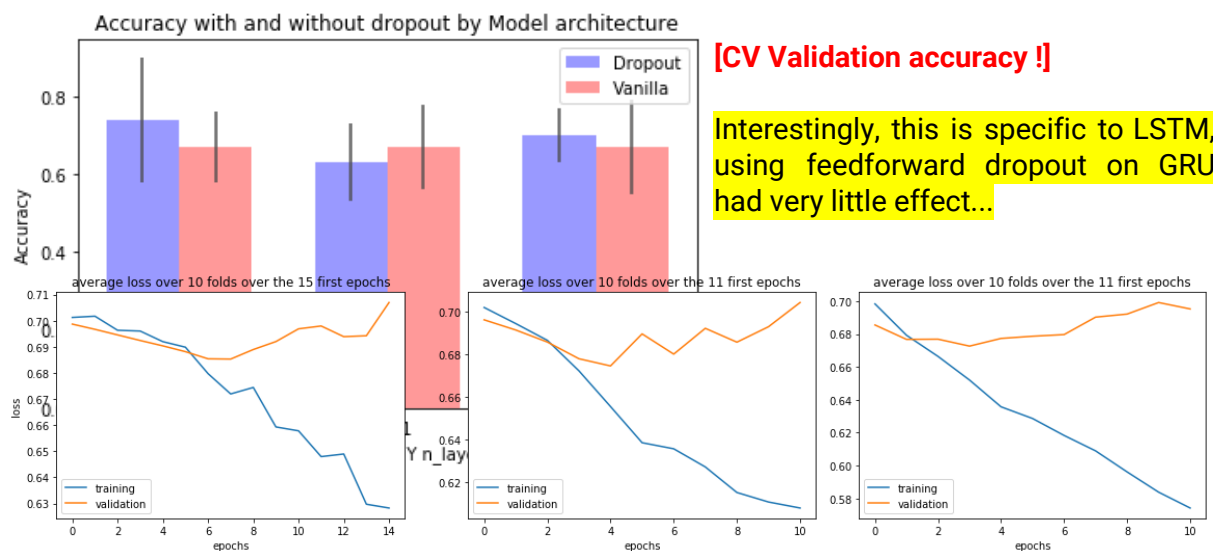


Fig. From left to right : 1GRU with and without dropout, bGRU without dropout (different scales)

4 Data

4.1 Exploration

After discarding subjects who didn't perform the *spiral* and task and counting from 0 :
 The tasks spiral, l, lektorka and tram from subjects 56, 9, 39 and 67, respectively, started their exam while pen was in air (i.e. not on paper). All these subjects are control except for n° 9.
 As you can see below it's neglectable for subjects 56 and 67 but leads to quite different plots for subjects 9 and 39.

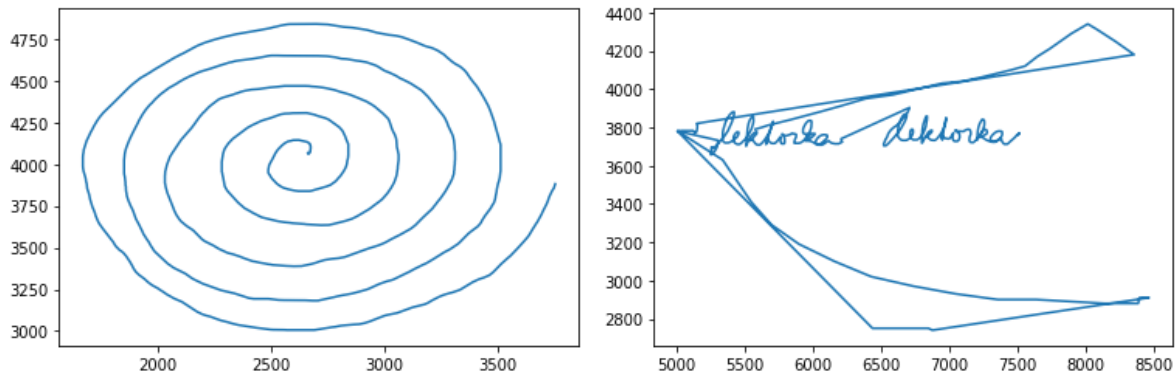


Fig. Exam starting in air. Left : *spiral* of subject n° 56, right : *lektorka* of subject 39. Both are controls.

In the same way the subject # 43 has a recording problem on his tram task : the timestamp measure jumps to 10^{12} on the 12 last points of the recording although the timestamps from all subjects are in the 10^6 magnitude. So we assume it's a recording error and discarded the 12 last points.

After investigation I found that most of the exams contained (really) small pauses. Therefore, the timestamp measure might be useful so the model has a sense of these pauses.

4.2 Preprocessing

Standardizing a vector (e.g. x-coordinate of a task) gives it a mean of 0 and a standard deviation of 1. Normalizing a vector gives it a norm of 1.

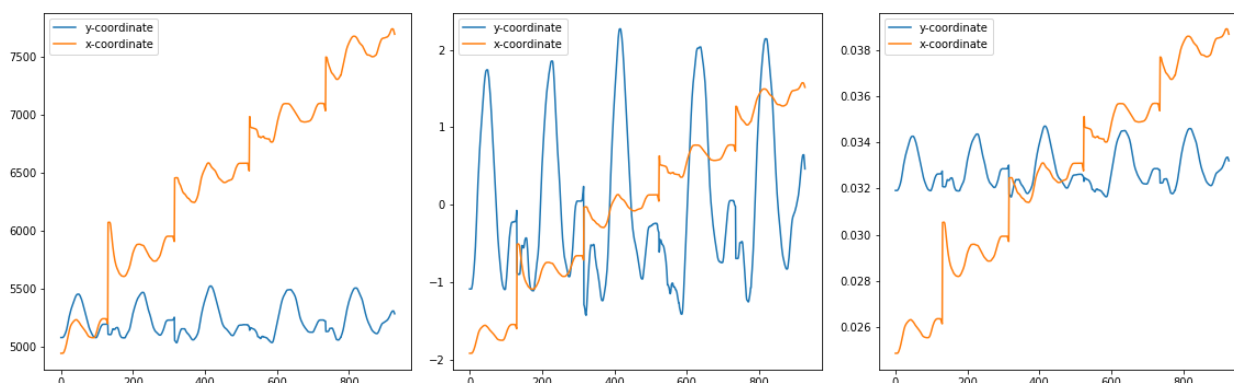


Figure of the x and y coordinate (for readability reasons) recorded at 200Hz for the second task (writing multiple "I") of the first subject (PD) : raw, standardized and normalized (from left to right).

One can notice that the data ranges are preserved when normalizing unlike when standardizing. So I'd assume that it was better to normalize than to standardize but after experiment, the model won't fit the data if it's normalized, although it fits it quite well if it's standardized. After investigation I noticed that, between subjects, the normalization shifted the values of the data : you can see here that for the first subject the normalized x coordinate ranges between 0.026 and 0.038, whereas for the 26th subject, it ranges from 0.022 to 0.028 (although the raw ranges are similar). If we feed the raw data to the model (i.e. without any type of normalisation) it doesn't fit to the data : with $lr = 10^{-2}$, the loss converges at 0.7 (accuracy 0.469). Tried with $lr = 10^{-3}$ (converges higher), $lr = 10^{-1}$ (doesn't converge).

Downsampling the data 10 times doesn't help with the overfitting as you can see in the experiment results (column downsampling factor).

Until April 30th, we naïvely standardized every measures, including the button_status which doesn't make any sense since it's a binary feature. Not standardizing the button_status slightly improved the results.

In the same way, we realized on May 10th that we standardized the timestamp which doesn't make much sense either as this measure primary gives us the speed or duration of the exam, information that is lost after independent standardization. Moreover, discarding the button_status and the timestamp provided similar results for the spiral task (since it's almost only on-paper strokes).

I don't know what is the unit of the timestamp. It's not a unix timestamp because interpreting it as is would mean that the exams have been recorded between 1970-01-02 and 1970-04-10 (over 4 months). Moreover, the exam duration (as unix timestamp) does not match the number of times steps times the sampling rate.

Therefore I removed t_0 from all timestamps so the measure would provide information about the duration and speed of the exam. Next came the standardization problem, as neural networks don't behave well with un-standard data (see above), I standardized the data

considering all measures at once instead of standardizing each subject's measures independently. I.e. I computed the average of all the x-coordinate instead of computing the average of the first subject's x-coordinate. This preserves the data ranges as you can see below.

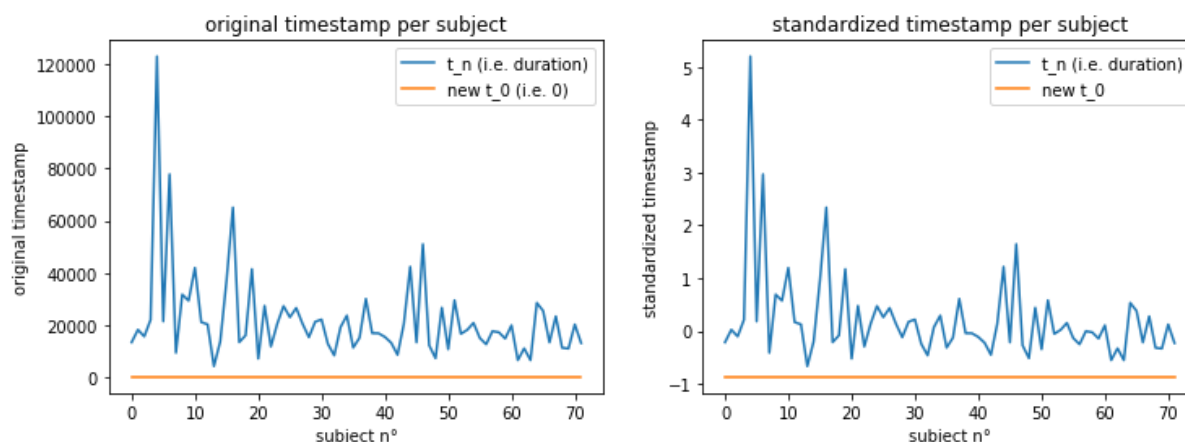


Fig. Timestamps per subject. Left : un-standardized timestamp (with t_0 removed). Right : globally standardized timestamps.

Strangely, with this global standardization, the RNN doesn't generalize at all and starts overfitting after 1 epoch. Thus the test accuracy doesn't go above chance level. This is maybe because the measures are not standardized per subject.

4.3 Representation

The RNN is *not* able to fit the data when training on all tasks at once. Although we're concatenating a one hot vector to the measures which identifies the task (e.g. [0., 1., 0., 0., 0., 0., 0., 0.] for the l task). This might be considered an early fusion technique which augments the dataset. We might want to compare it to late fusion techniques (e.g. majority voting).

In order to be able to implement *attention* (cf. Report #4 [1.1 Model Architectures et al.](#)) I tried trimming and padding with zeros the spirals that were respectively longer and shorter than 3117 timesteps (3rd quartile of the spirals' length) and the RNN won't fit the data with the same hyperparameters as without trimming and padding.

Inspired by the work of Zhang et al. (cf. report #5). I tried to split the data into subsequence with a fixed time window of 100 timesteps. Unfortunately, the RNN is completely unable to generalize and the validation accuracy never gets better than chance level, although the RNN fits the training set. Dropout was ineffective. **Update this might also be because the windows are not standardized, the whole task is standardized.**

I should maybe try other segmentations, e.g. stroke, also I might want to train 2 separate RNNs on in-air and on-paper strokes, respectively. This is motivated by the experiment results which are superior on the *spiral* task where the single stroke is on-paper. As opposed

to the *l* task where strokes are both on paper and in-air. Since in-air movements are very different from on-paper ones, I think they might disturb the RNN.

Interestingly, computing the movement (cf. report #5) had no effect when the data was split in subsequences but improved the results when training on the whole sequence, although Leo told me it shouldn't be necessary when using a RNN, therefore, further experiments are required.

I tried to split the *l* task into strokes then to train 2 different encoders :

- one on the in_air strokes and
- one on the on_paper ones.

I tried to use 2 separate decoders as well as 1 shared. The predictions are then averaged over each subject to get a final prediction (as in the fixed-time window experiment). The results are not encouraging and similar to "whole sequence"-training on the *l* task : we get around 70% accuracy on the validation set with very high std (between 15% and 20%) and we barely get above chance level on the test set.

When using a single RNN over the strokes (thus learning both in_air and on_paper strokes simultaneously) the test accuracy falls behind chance level. Even though we feed the button_status unscaled thus indicating whether the stroke is in_air or on_paper.

Update this might also be because the strokes are not standardized, the whole task is standardized.

In view of using lam-onDB for transfer learning (cf. report #5), I tried to fit the GRU-baseline model using only the x and y coordinates. The model fits the data but the test accuracy barely goes beyond chance level (no early stopping).

I also tried to discard the button_status and the timestamp measures, as I assumed that they were not interesting for the spiral task. The results are quite similar with the GRU-baseline, slightly inferior (no early stopping).

4.4 Augmentation

I tried 5 different heuristics of data augmentation : 4 applicable only to spatial coordinates : flips, rotations, translation and homothetic (i.e. "zoom"), 1 applicable to all measures : gaussian noise. I made the assumption that gaussian noise would be ineffective since PD suffer from tremor which translates into a "noisy" signal. Moreover, Graves 2012 (cf. Report #4) argue that Gaussian input noise is only effective if it mimics the true variations found in the data, e.g. for speech recordings.

All augmentations were applied both to the training and the validation set because one of the main goals was to augment the validation set in order to have a better representation of the model generalization capacity to perform early stopping (cf. [5 Evaluation and experiments](#)).

In order to have a dataset size of the same order between these 5 heuristics, I applied 3 transformations every time (cf. figures below), thus multiplying the number of training examples by 4.

None of these augmentation heuristics provided encouraging results, although I only tried with the GRU-baseline.

4.4.1 Flips

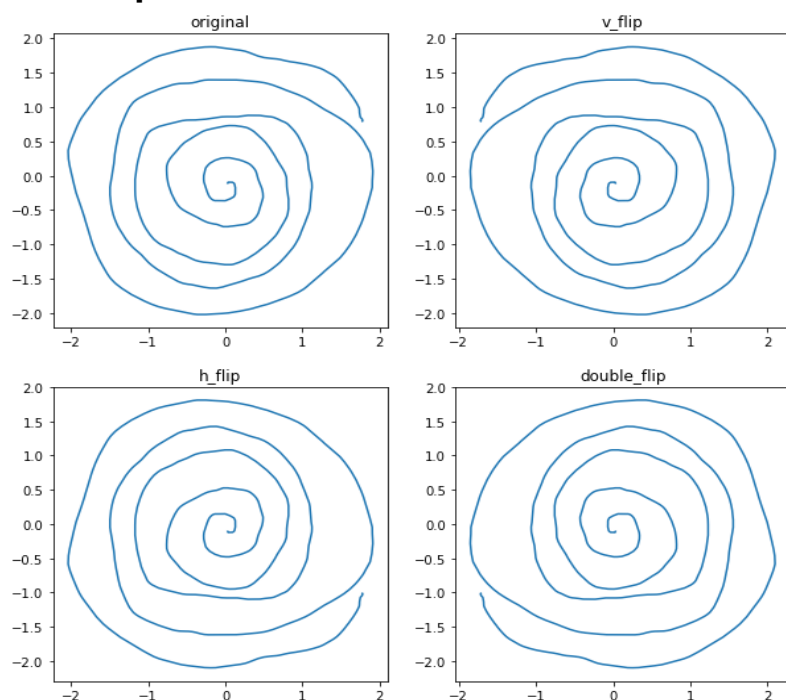


Fig. Flips : original data, vertical flip, horizontal flip, and both horizontal and vertical flip

For the flips I empirically choose a symmetry axis along which to flip the data, e.g. for the vertical flip, I transposed every point with respect to the vertical axis (i.e. the first x point).

4.4.2 Rotations

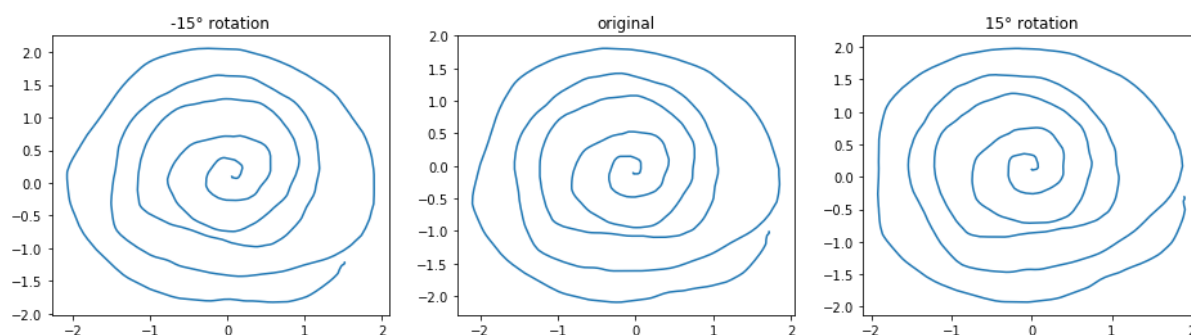


Fig. Rotations. Data is re-centered after rotation.

To have a dataset size of the same order as with flips I performed a third rotation of $+30^\circ$.

4.4.3 Translations

Before each training step I choose a random number between -0.5 and 0.5 (as the standard deviation of the data is one). I then apply 3 different types of translations :

1. translation along the x-axis
2. translation along the y-axis
3. translation along both x-axis and y-axis

4.4.4 Homothetic (i.e. "zoom")

An homothety is a linear transformation where we multiply a vector by a scalar. Therefore when the vector is a spatial coordinates we "zoom" or "de-zoom" the image when the scalar is greater and smaller than 1, respectively. In order to have data ranges comparable to the translation, I choose a random zooming factor between 0.8 and 1.2, as the standardized spirals coordinates ranges from ~ -2 to ~ 2 and the translation ranges in $[-0.5, 0.5]$. As for the translation, I performed 3 types of homothety :

1. homothety along the x-axis
2. homothety along the y-axis
3. homothety along both x-axis and y-axis

4.4.5 Gaussian noise

I choose to add gaussian noise on all the measures with standard deviation of 10^{-2} as it's almost invisible for the naked eye on the pressure. With $\text{std} = 10^{-3}$, the variations of the spatial coordinates are also invisible for the naked eye. I called this experiment "naïve noise" because I add gaussian noise on all measures, even timestamp and button_status.

I conducted another experiment with noise only on noisy measures. Namely, tilt, elevation and pressure (which are noisy independently of PD and controls). I called it "noise on noisy". It provided better results than naïve noise but didn't improve the baseline (without augmentation).

In order to have a data size of the same order as the previous heuristics, for each data sample, I made 3 random noisy variants when training and validating, thus multiplying the data size by 4.

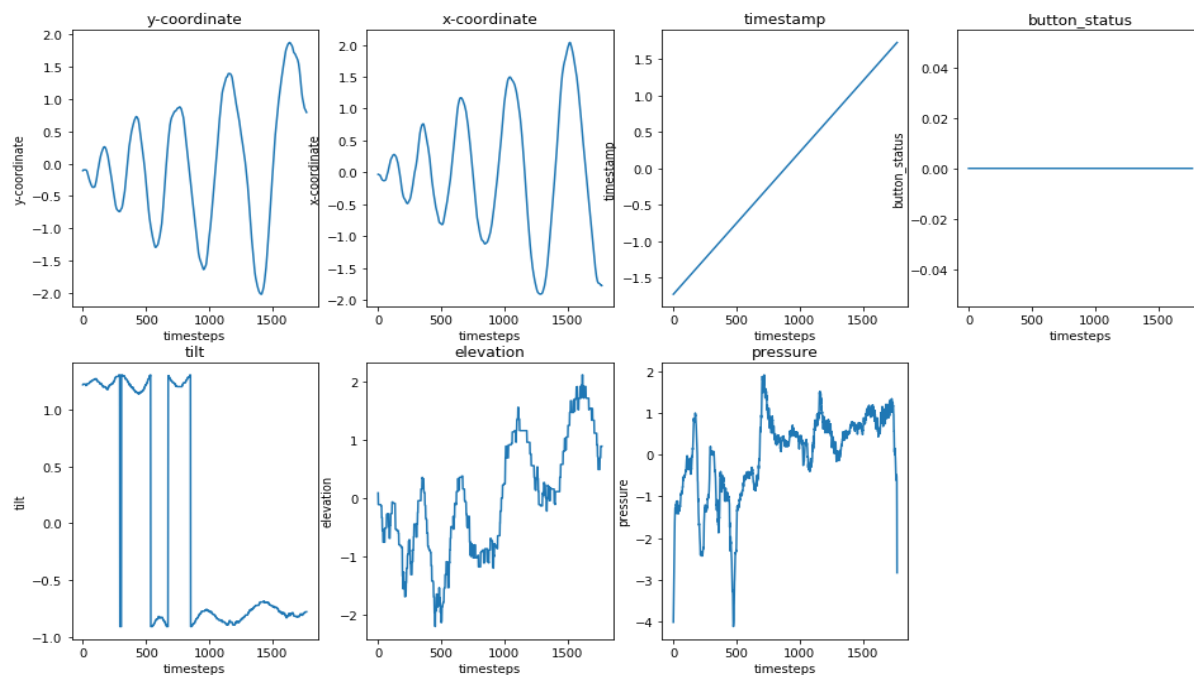


Fig. All 7 measures plotted against time for the first subject spiral. We can see that tilt, elevation and pressure are noisy unlike spatial coordinates, timestamp and button_status.

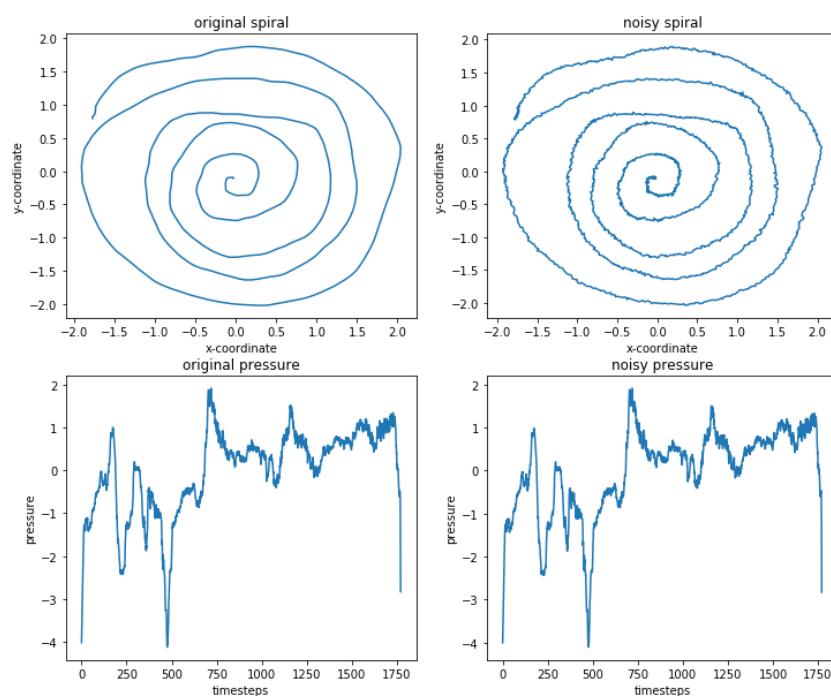


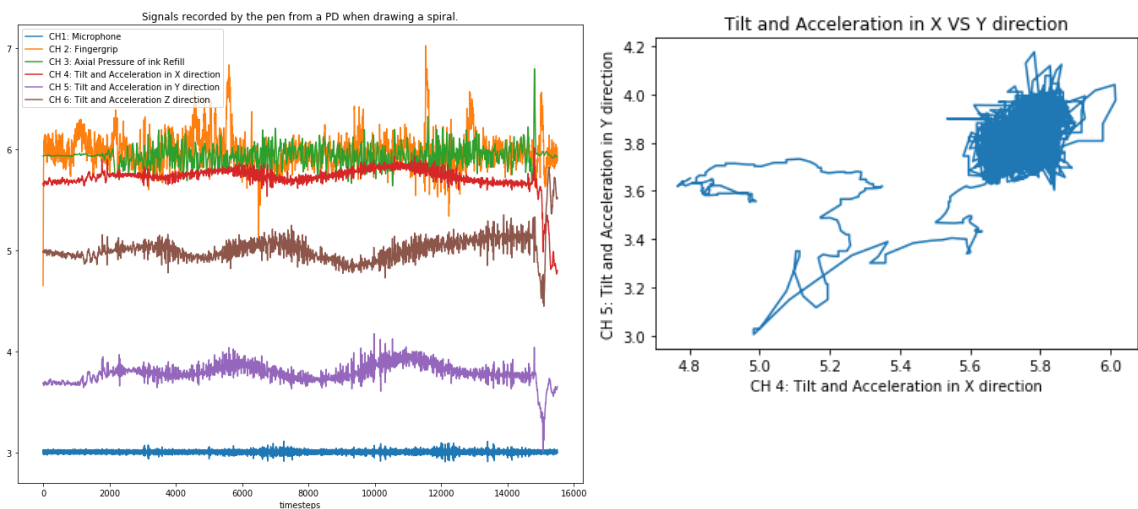
Fig. First subject spiral, with (right) and without (left) gaussian noise with 10^{-2} std.

4.5 NewHandPD

of the dataset.

See Report #6 for analysis

Hoping to achieve transfer learning I took a look at the NewHandPD database of Pereira et al. (cf. Report #2). However I'm not sure we'll be able to achieve transfer learning between NewHandPD and PaHaW as NewHandPD doesn't record the x and y coordinate but the "tiltand acceleration" in both x and y directions.



Thus the plot of X vs Y is not visually interpretable.

5 Evaluation and experiments



Fig. 10 fold Cross-validation (CV) configuration until April 25th. The numbers represent the number of subjects in each fold. Training set is colored in blue while the validation/test set is colored in green.

Average metric over the 10 folds (\pm standard deviation) after early stopping (i.e. we select the best epoch based on the validation accuracy for each fold). For the baseline model depicted in [1 Model architecture](#).

TRAIN accuracy	accuracy	Se	Sp	PPV	NPV
0.59 (+ 0.07)	0.68 (+ 0.09)	0.54 (+ 0.20)	0.82 (+ 0.20)	0.81 (+ 0.20)	0.65 (+ 0.08)

[Validation accuracy !]

Every metric is for the validation set if not specified otherwise (i.e. except for the first column). Since the validation set is very small (~ 7 subjects) the metric standard deviation (std) is very high, this is consistent with Moetesum et al. (cf. Report #2). The accuracy for the *spiral* task is 76%, 63% and 55% for Moetesum et al., Drotar et al. and Impedovo et al.,

respectively. Notice that, in the same way as Drotar et al., we validate on the test set (i.e. on the validation fold). Therefore our results are over optimistic and possibly overfitted to the dataset.

I tried to train the model without early stopping for 4 epochs (because the model usually overfits after 4 epochs) with the GRU-baseline. The accuracy is slightly decreased and the std augments : from 70% to 65% and from 8% to 13% respectively. Moreover, these results are also "overfitted" to the test set because if we train for one more epoch they get worse so I didn't present them.

I also tried to implement a proper early stopping by splitting the dataset in training, validation and test set and early stopping according to the validation results. However in this case the results are even worse, only 61% accuracy and 17% std.

I don't think the problem is that the training set is too small because if we take the best epoch for each test folds we get the same results as before : 71% ($\pm 11\%$) accuracy. The problem is that since the validation test is very small, it's not very representative of the generalization capacities of the model. On average the validation loss and the test loss are quite similar (cf. figure below) but on some folds they're very different. We might try to do data augmentation on the validation set so it's bigger and thus more representative of the capacities of the model.

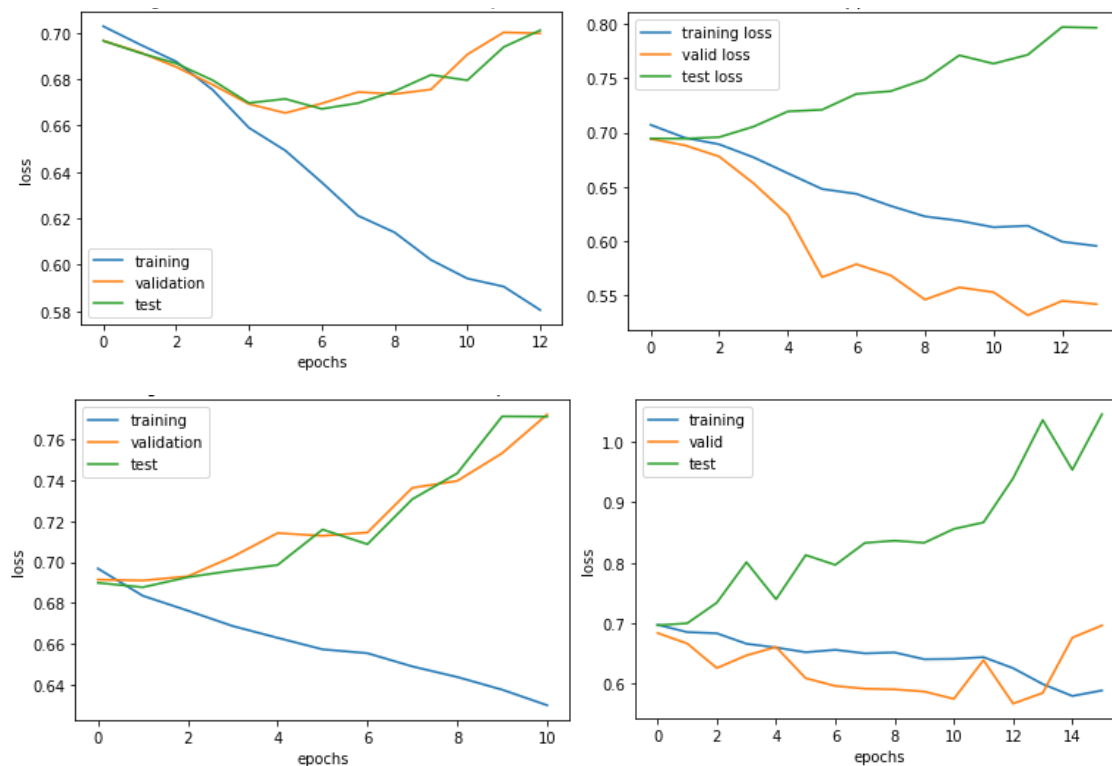


Fig. Left : average loss over the 10 folds, right : loss on the first fold. First row : no data augmentation. 2nd row : Flips (similar with rotations, homothety, naïve noise and translations).

8	8	8	8	8	8	8	8	6	6
---	---	---	---	---	---	---	---	---	---

Fig. 10 fold Cross-validation (CV) configuration after April 25th. The numbers represent the number of subjects in each fold. Training, validation and test sets are colored in blue, orange and green, respectively.

Even though these results are disappointing, I think we should stick with one of these methods because Drotar et al. validate on the test set for hyperparameters search (as us), not for the n° of epochs etc. I hope we will be able to conjecture from the previous experiments results which were over optimistic. As the data augmentation did not improve the results (cf. [4.4 Augmentation](#)), I think we should not perform early stopping and train for fixed n° of epochs.

Following the guidelines of Greff et al. (cf. Report #4 [1.2 Hyperparameters finetuning](#)), we should be able to fine tune the hyperparameters independently, starting from :

- learning rate
- hidden size
- number of layers
- regularization techniques (e.g. dropout, although Karpathy² and Lipton et al. suggests that it interacts with the hidden size so we might want to tune these hyperparameters together)

Throughout these experiments we shall use a bidirectional LSTM, which can only better the results in my opinion (cf. [1 Model architecture](#)). Also the decoder architecture will be kept as simple as possible (cf. [1 Model architecture](#)). I made this choice because I assume that the "hard part of the job" is made by the LSTM and not the decoder.

I thought we might also use an SVM as Moetesum et al. or OPF as Passos et al. 2018 (cf. Report #2) but we can't because unlike them we need to train the network (they use a CNN pretrained on ImageNet).

All the results of the experiments are available in the git repo. The studied hyperparameter is print in **bold**.

Note :

- the max epochs was set at 5 for the lines 20 and 21, 10 for the lines 22 and 23 and 50 for the rest
- there was no patience for early stopping for the lines 20 to to 26 but no improvement was okay : you can see the differences between line 26 (which is greyed out) and 27. They are probably explained because when we tolerated the lack of improvement, the model could wait for tens of epochs (cf. "early stopped"), overfitting until "by chance" the accuracy would went up (cf. figure below : left for accuracy, right for loss). Thus the **results of lines 26 are overoptimistic**.
- the seed for random weights initialization was not fixed for the lines 20 to 26. Therefore a direct comparison with and between those might be biased.

² <https://github.com/karpathy/char-rnn>

- Some "falses" are greyed out, it's because they represent subject indexes before discarding the 3 subjects who didn't perform the spiral task. Therefore these index don't match the other ones.

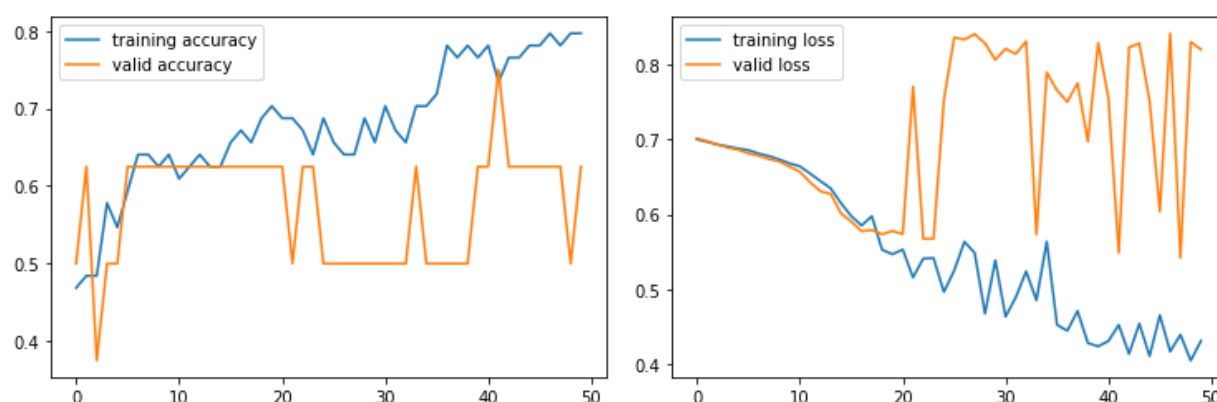


Fig. Model overfitting until (probably) random amelioration (line 8). Left : accuracy, right : loss.

6 Visualization & Interpretation

6.1 Input weights

Karpathy, Johnson et al. wrote a great article about visualizing RNN, however, their classification task (language modeling) allows for a simpler interpretation. I tried visualizing the input weights of a model fitted to the data in order to see which of the 7 measures was activated. As you can see below, the weights differ along the layer (i.e. between the GRU units) without any measure standing out (I also tried plotting each measure along the layer or compute the mean of each measure). One possible interpretation is that there's no measure which is consistently "interesting".

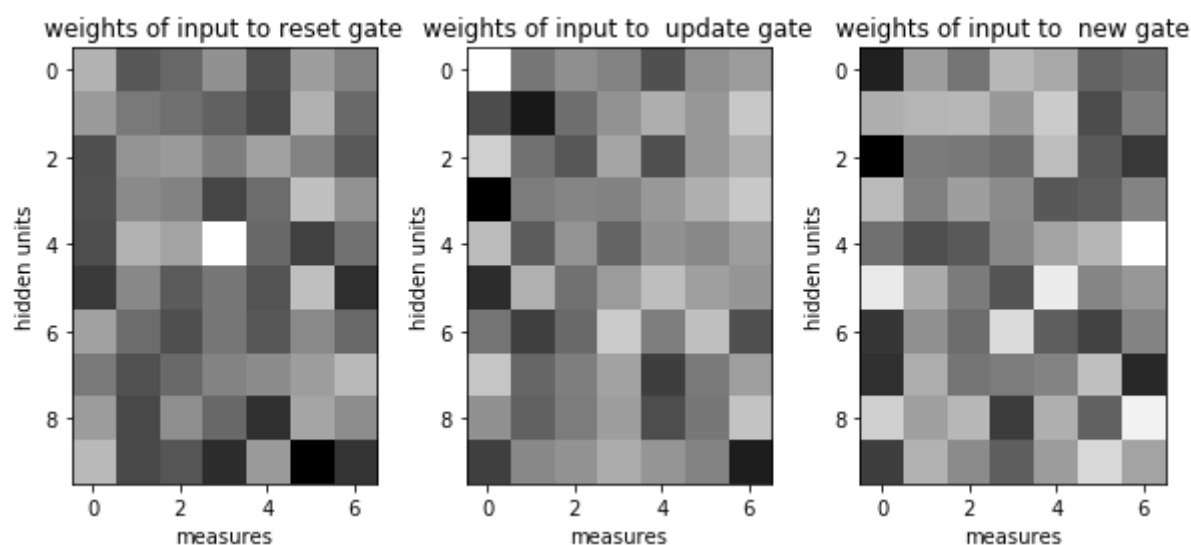
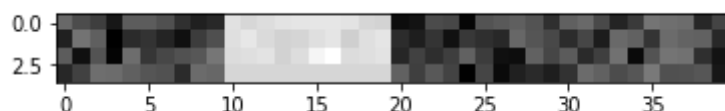


Fig. Plot of the weights of the hidden units (of the first layer of the network).

6.2 Forget Gate

Interestingly, after using the trick of init the forget gate bias at 1, even after 22 epochs and an overfitted model, the bias of the forget gate are still worth ~ 1 (cf. figure below). One possible interpretation is that the model learned to *remember* the previous value of the *cell state*.



6.3 Misclassified subjects

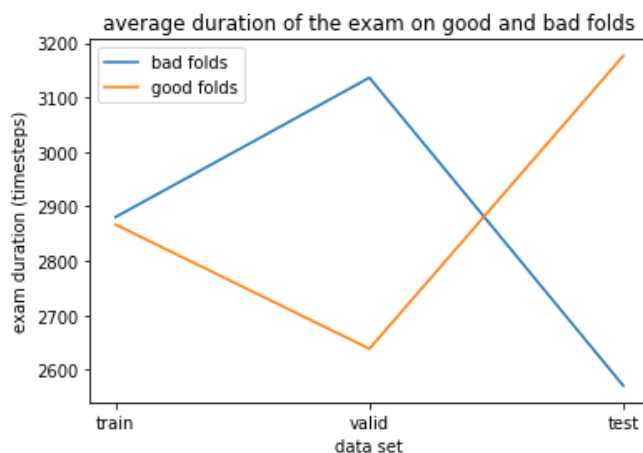
For the RNN with hyperparameters :

dowsam pling factor	learning_ rate	hidden_s ize	num_lay ers	bidirectio nal	carry over	dropout	gradient clipping
1	0.001	50	1	TRUE	0	0.5	5

I looked into which subjects had been misclassified when the model had early stopped (i.e. when the classification accuracy was highest). It turns out that, among the misclassified subjects (22 over all 10 folds), 3 of them had performed very long exams of 16071, 9724, and 8581 timesteps. Thus bringing the average length of misclassified subjects to 3552 instead of 2758 (average over every subjects). It'd be interesting to see if the same subjects are misclassified across models.

On the GRU-baseline, on the contrary, I found that misclassified subjects had performed in average an exam of 2.5k timesteps, thus lower than average. This might be because this experiment was done after the global standardization of the timestamps (cf. [4.2 Preprocessing](#)) thus giving a sense to the model of the duration of the exam.

Moreover, I looked into the performance difference between folds and classified as a "good fold" one where the test loss didn't explode before the 10 first epochs. And as a "bad fold" otherwise. As the 10 CV is stratified, the distribution of classes is perfectly distributed between good and bad folds. Moreover, I looked into the size of the different training and test sets among good and bad folds and didn't find any difference either. However, I found that subjects of the test set had a an average duration significantly higher on good folds than on bad folds (see figure below). While the duration on the training set is similar in both good folds and bad folds. We could conjecture from that that it's easier to classify long exam : maybe because the model has more timesteps to make a decision ? One might think that it's because some PDs have longer exams but, again, the sets are equally distributed and, moreover, the plot is very similar when looking only at controls duration or PDs duration (i.e. both control and PD contribute equally to the longer duration on good folds).



Additionally, we can see on the plot that the average duration is very different from the validation set to the test set, this might explain why early stopping is not effective (i.e. why the validation set performance does not represent the overall performance of the model.)

Conclusion - Todo

List

I followed my intuition and explored the hyperparameters fine-tuning of our current model on the *spiral* task. I found that GRUs were less prone to overfitting than LSTMs, although feedforward dropout was efficient for regularizing LSTM. Thus, stacked LSTMs give better results than long, single layers. In the experiment results, I colored in green the hyperparameters that worked well together, as well as all the learning rates and gradient clipping which I found the best values to be 10^{-3} and 5, respectively. Again, all the experiments with "proper_early_stopping = False" were conducted while validating on the test set (cf. [5 Evaluation and experiments](#)) so I hope the results are still valid when running a proper CV.

Then I tried, without success, to learn from all tasks at the same time, then to learn from fixed-time subsequences (cf. [4.3 Representation](#)).

Data augmentation didn't provide encouraging results (cf. [4.4 Augmentation](#)) so I think we should not perform early stopping and we might consider using a larger dataset or change model.

Also, I think my work on [4.5 NewHandPD](#) is interesting and might deserve a paper.

Overwrites report #4 todo list.

1. ask san luciano's dataset (10 spirals with each hand of 138+150 subjects = 5760 total samples) : no answer after 3 weeks...
2. architectures of report #4 [1.1 Architectures I want to implement](#) : partly done, see [1 Model architecture](#).
3. 1D CNN : done, see report on CNNs.
4. advanced transformations (cf. report #3), see report on CNNs :
 - a. dynamic time warping (maybe not efficient alone, better to use along with hand-crafted features)
 - b. discrete wavelet transform (may not be efficient as Afonso et al. only achieved 83% accuracy on newhandpd)
 - c. power spectral density (PSD)
5. train a model on each task and merge prediction (i.e. late fusion)
6. regularization of report #4 [3 Regularization & Overfitting](#) (e.g. recurrent dropout) : not relevant if we use CNN
7. auto encoder of report #4 [4 Transfer Learning](#)

References

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.